Performance Troubleshooting in Data Centers: An Annotated Bibliography*

Chengwei Wang[1*], Soila P. Kavulya[2†], Jiaqi Tan[2], Liting Hu[1], Mahendra Kutare[3], Mike Kasick[2], Karsten Schwan[1], Priya Narasimhan[2], Rajeev Gandhi[2]

[1]Georgia Institute of Technology [2]Carnegie Mellon University [3]Boundary, Inc.

1. INTRODUCTION

In the emerging cloud computing era, enterprise data centers host a plethora of web services and applications, including those for e-Commerce, distributed multimedia, and social networks, which jointly, serve many aspects of our daily lives and business. For such applications, lack of availability, reliability, or responsiveness can lead to extensive losses. For instance, on June 29^{th} 2010, Amazon.com experienced three hours of intermittent performance problems as the normally reliable website took minutes to load items, and searches came back without product links. Customers were also unable to place orders. Based on their 2010 quarterly revenues, such downtime could cost Amazon up to $1.75 million per hour, thus making rapid problem resolution critical to its business. In another serious incident, on July 7^{th}, 2010, DBS bank in Singapore suffered a 7-hour outage which crippled its Internet banking systems, and disrupted other consumer banking services, including automated teller machines, credit card and NETS payments. The cascading failure occurred due to a procedural error while replacing a faulty component in one of the bank's storage systems that was connected to its main computers.

The high-cost of downtime in large-scale distributed systems drives the need for troubleshooting tools that can quickly detect problems and point system administrators to potential solutions. The increasing size and complexity of enterprise applications, coupled with the large scale of data centers in which they operate, make troubleshooting extremely challenging. Problems can arise due to a large variety of root-causes because of the complex interactions between hardware and software systems. The large volume of monitoring data available in these systems can obscure the root-cause of these problems. Lastly, the multi-tier nature of applications composed of entirely different subsystems managed by different teams complicates problem diagnosis.

The importance of troubleshooting has given rise to many systems and methods deployed in industry and developed in research. The purpose of this bibliography is to summarize the state of art by identifying its key characteristics and contributions, and pointing to opportunities for future work. We focus on performance troubleshooting, rather than on the general topic of failures in distributed systems, by addressing the following topics: terminology (Section 2), challenges and opportunities (Section 3), troubleshooting methodology (Section 4), detection (Section 5), diagnosis (Section 6), supporting infrastructures (Section 7), and representative remediation methods (Section 8).

Topics out of scope for this bibliography include the rich set of research on reliability and high availability in distributed systems [1, 2], monitoring for parallel and high performance systems [3], autonomic system management and adaptation [4, 5], debugging [6] and fault management or understanding [7], and many other topics pertaining to the construction and maintenance of efficient and highly-available distributed systems and applications.

[1] A. M. Johnson, Jr. and M. Malek. Survey of Software Tools for Evaluating Reliability, Availability, and Serviceability. *ACM Computing Surveys*, 1988

[2] D. Long, A. Muir, and R. Golding. A Longitudinal Survey of Internet Host Reliability. In *Proceedings of the 14th Symposium on Reliable Distributed Systems*, 1995

[3] C. L. Mendes and D. A. Reed. Monitoring Large Systems via Statistical Sampling. *International Journal of High Performance Computing Applications*, 2004

[4] M. C. Huebscher and J. A. McCann. A Survey of Autonomic Computing – Degrees, Models, and Applications. *ACM Computing Surveys*, 2008

[5] J. O. Kephart. Autonomic Computing: the First Decade. In *ACM International Conference on Automatic Computing (ICAC)*, 2011

[6] C. E. McDowell and D. P. Helmbold. Debugging Concurrent Programs. *ACM Computing Surveys*, 1989

[7] F. C. Gärtner. Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments. *ACM Computing Surveys*, 1999

2. TERMINOLOGY

A *data center* is a company's facility housing servers and associated components, such as networking equipment, storage systems, cooling systems, and power supplies. The number of servers in a data center can be large. Unoffi-

*this work was funded in part by the Intel Science and Technology Center for Cloud Computing (ISTC-CC)

†Chengwei Wang and Soila P. Kavulya contributed equally to this literature review project

cially, it is estimated to a range from more than 1000 to over 1 million[1]. Data centers can be consolidated or/and non-consolidated, where consolidated data centers use virtualization technology to run virtual machines (VMs) on physical servers and deploy applications in VMs, while non-consolidated data centers deploy applications on physical servers. Hybrid data centers use both VMs and bare servers to host applications. Typically, the applications running in these systems are server-side applications, such as traditional web services, data analytics, and multimedia streaming services. They may belong to a single enterprise or to multiple parties, since servers or VMs can be rented to customers as managed data centers and/or cloud services.

Depending on management needs and service objectives, *performance* can be defined at different levels of abstraction — application, VM, OS, hardware, and from different viewpoints — application user, infrastructure provider, etc.. Performance can also be described for an individual server (e.g., the server's CPU utilization, memory utilization) or for a set of servers as aggregate metrics (e.g., ranked power usage, application level request latency of a web service hosted by a number of servers). However, in daily data center operations, there are typically a small number of metrics of importance to a company's business interests. These metrics, termed *key performance indicators — KPIs —* are commonly used as measurements describing high-level performance objectives. Each data center management scenario may have a unique set of KPIs.

We define *performance problems* as violations of KPIs and/or issues (failures, errors, resource contention, or other abnormalities) that contribute to KPI violations. In practice, for each user and provider, there is a relatively small number of KPIs, but there is an extensive set of low-level measurements for each problem, and operator definitions/perceptions of such problems vary.

Performance problems occur due to a large variety of root causes. Faults can occur at each level of the data center — from transient or permanent hardware faults, to unforeseen component interactions, to software misconfiguration or bugs. This bibliography focuses on only those methods and systems that address *performance troubleshooting*, which can be defined as a procedure that (1) detects a performance problem, (2) diagnoses the problem to the extent needed to remedy it, mitigate its effects, or even determine its root cause, and (3) applies remediation activities to restore the system back into correctly operating state.

3. CHALLENGES AND OPPORTUNITIES

The following characteristics of enterprise applications and data centers challenge performance troubleshooting.

1. Scale. It is not unusual for a data center to have thousands of servers. Large web companies' data centers can have over 1 million servers. In consolidated data centers, each server can host hundreds of VMs, and each VM hosts hundreds of application processes supporting various interacting components of application services. At this scale, performance problems occur frequently, and in response, software in these systems is written to deal with potential sources of problems through built-in error logging and tracing. With large volumes of monitoring/tracing data, troubleshooting is much like finding a needle in a haystack.

2. Complexity. Performance troubleshooting in the massively distributed environment of today's data centers goes beyond what is done in private data centers that only host applications owned by a single company. (1) Applications commonly consist of distributed software components deployed on different machines or even different data center sites. (2) Components may come from different software vendors or open-source developers. (3) Component interactions are complex, not only due to scale, but also because they use built-in resilience methods like those based on replication, quorums, and automatic restart. (4) Different teams of developers may be responsible for the different services or tiers of SOA-based applications; therefore, multiple teams need to work together when something goes wrong with a service. (5) Public clouds, like Amazon EC2 or Google App Engine, experience greater levels of complexity than private data centers because they host applications from a diverse set of customers. Data-center operators typically have little knowledge about the implementation logic — one reason being to protect customer privacy. Conversely, for security and compliance reasons, application developers may not have direct access to underlying hardware — requiring them to request operations teams to provide them with the relevant log and trace files for troubleshooting. If analysis of one server's log fails to reveal the problem because a different server is responsible for the fault, the manual, tedious and error-prone process must be repeated.

3. Dynamism. A data center is a shared infrastructure, with frequently changing users, and with applications frequently installed/deployed and removed. Workloads vary over time, *i.e.*, *temporally*, and across data center nodes, *i.e.*, *spatially*. These variations are exacerbated by their aforementioned resilience methods. The situation is even worse in virtualized public clouds, where VMs running a large variety of customer applications can be created, migrated and terminated dynamically.

Scale, complexity, and dynamism make it extremely difficult to diagnose data center problems, yet effective performance troubleshooting remains crucial for both data center users and providers, because data center providers seek:

- increased hardware utilization and reduced resource consumption,
- reduced IT costs,
- consolidation with automated management tools, all
- to optimize their investment at cloud scale.

while users demand:

- increased service availability and reliability,
- increased service performance and productivity,
- more proactive response to customer needs,
- less downtime for increased revenue, all with
- appropriate levels of security and data privacy.

4. METHODOLOGY

Performance troubleshooting in enterprise date centers typically involves the following phases.

1. Detection is the phase that identifies violations of KPIs or anomalies that may lead to KPI violations. The outcomes

[1]Companies like Google don't officially reveal the size of their data centers. Estimates are based on calculations of data center energy consumption, etc.

of detection includes the time(s) when the problem is detected and the metrics or KPIs related to the problem. With KPIs that denote end-to-end performance, detection of KPI violations is often too late because performance degradation has already affected user experiences. These violations also typically lead to loss in revenue and/or system downtime. Furthermore, KPIs are hard to define in some circumstances, particularly in public clouds, as a cloud users' KPI may be unknown. Research has responded to these issues by moving from directly measuring KPI violations to detecting candidate anomalies before such violations occur, and as a result, anomaly detection has become an important area of research in performance troubleshooting. We will discuss this research in Section 5.

2. Diagnosis is the phase in which KPI violations or anomalies are analyzed to determine their causes or/and suggest solutions. One of the outcomes of diagnosis phase is the *location* of the problem, which enables subsequent mitigation or remediation. However, identifying root-causes and fixing them typically require human involvement and/or the use of sophisticated problem-solving tools that typically require domain expertise to make intelligent inferences.

3. Remediation or mitigation is the phase of taking actions to recover the system to normal state or to prevent it staying in an abnormal state. The result of remediation can validate the detection and diagnosis phases, making it possible to create a 'closed loop' for automated performance troubleshooting.

Though there is a few research in which there is no clear border between detection and diagnosis phases, we follow the organization suggested above when categorizing existing literature, for clarity sake. We also classify the systems and infrastructures that support performance troubleshooting as follows.

1. Monitoring systems collect and aggregate metrics from the systems they observe. Monitoring systems aim to provide a global view. They typically compile summary statistics to facilitate continuous monitoring in large scale applications and systems.

2. Tracing systems aims to provide detailed information by collecting targeted events from the systems being observed, such as the messages being sent and received, system calls executed, *etc.* Tracing systems may also track certain system behaviors, like the call patterns or service call paths present in distributed applications. Tracing can impose considerable overheads on target systems, with large trace logs stored in files. Tracing has recently gained additional importance due to the widespread use of instrumentation, logging user and/or system events by web companies able to monetize such data.

3. Analytics systems support analysis and queries, based on monitoring and/or tracing data acquired about the targets under observation. They may operate online — as their target applications run, offline — based on previously collected monitoring or log data, or both, sometimes also termed as operating on 'data in motion' vs. 'data at rest'.

5. DETECTION

Problem detection algorithms can be classified into two categories: (1) *reactive approaches* that detect anomalies after they occur, and (2) *proactive approaches* that predict impending anomalies when the system state is still normal. The reactive approach does not incur any *prevention* cost but can lead to prolonged service downtime — which can be prohibitive for real-time applications like online web logs, live sensor stream processing [9], and many business applications [10]. In contrast, the proactive approach offers better responsiveness, but at the risk of imposing continuous overhead for prediction and possible penalties for inaccurate detection of anomalies — false alarms. We will follow this categorization when detailing existing approaches in Section 5.2 and Section 5.1.

[9] C. Wang, I. A. Rayan, and K. Schwan. Faster, Larger, Easier: Reining Real-Time Big Data Processing in Cloud. In *ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, 2012

[10] A. Gavrilovska, K. Schwan, and V. Oleson. A Practical Approach for 'Zero' Downtime in an Operational Information System. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2002

5.1 Reactive Detection

Reactive approaches aim to detect problematic states of data center systems or applications. Data center users are concerned with external states, expressed by KPIs. Data center operators are concerned with internal states — their daily jobs being to handle abnormal internal states. However, abnormal states are difficult to describe with simple metrics and associated thresholds. This is due, in part, to system scale and complexity; but a more important reason is that performance problems in the data center may only manifest at application level. An instance reported in [11] is a web site that correctly responds to pings, HTTP requests, and returns valid HTML; yet, at the application-level, returns only partially filled web pages or incorrect page content. These problems are difficult for data center operators to detect because they have limited knowledge about applications. These difficulties are compounded by the diversity of data center software and hardware provided and maintained by multiple vendors or organizations.

[11] E. Kiciman and A. Fox. Detecting Application-Level Failures in Component-based Internet Services. *IEEE Trans. on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, 2005

Problem detection methods are designed from many different perspectives. From the knowledge base perspective, MASF (Multivariate Adaptive Statistical Filtering) [12] and WISE [13] are *threshold-based* algorithms setting up thresholds of metric values along with the time pattern used for monitoring. They detect performance problems as threshold violations on individual observations [12] or on a series of observations [13]. Threshold-based approaches typically cater for seasonal variations in workload, *e.g.*, heavier load during the week and lighter load over the weekend, by using different thresholds for different seasons. The main challenge with these techniques is selecting a threshold that maximizes true positives while minimizing false positives.

[12] J. P. Buzen and A. W. Shum. MASF – Multivariate Adaptive Statistical Filtering. In *Computer Measurement Group (CMG)*, 1995

[13] S. Meng, T. Wang, and L. Liu. Monitoring Continuous State Violation in Datacenters: Exploring the Time Dimension. In *IEEE International Conference on Data Engineering (ICDE)*, 2010

Pinpoint [11], Console Log Analysis [14], SPA (System Performance Advisor) [15] and the profile-driven model [16] are *modeling-based* approaches that use machine learning models [11, 14], time-series models [15] and queuing models [15, 16] to describe computer system performance, where performance problems are recognized by deviations from these models. Model-based approaches are well-suited for detecting application-level problems. However, building models requires an in-depth understanding of the system.

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009
[15] P. Hoogenboom and J. Lepreau. Computer System Performance Problem Detection Using Time Series Models. In *USENIX Annual Technical Conference (ATC)*, 1993
[16] C. Stewart and K. Shen. Performance Modeling and System Management for Multi-component Online Services. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005

Flow Intensity [17], PeerWatch [18], Reference-Execution [19], Metric-Correlation Models [20] and Invariants Mining [21] are *correlation-based* approaches. Correlation-based approaches learn normal behavior by analyzing historical data to automatically discover correlations between metrics that are stable over time. They assume that performance problems break these correlations. Care needs to be taken when selecting thresholds for determining whether a correlation is significant — if the thresholds are too low, these approaches may discover spurious relationships which result in increased false positives.

[17] G. Jiang, H. Chen, and K. Yoshihira. Discovering Likely Invariants of Distributed Transaction Systems for Autonomic System Management. *Cluster Computing (Springer)*, 2006
[18] H. Kang, H. Chen, and G. Jiang. PeerWatch: a Fault Detection and Diagnosis Tool for Virtualized Consolidation Systems. In *ACM International Conference on Automatic Computing (ICAC)*, 2010
[19] K. Shen, C. Stewart, C. Li, and X. Li. Reference-Driven Performance Anomaly Identification. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2009
[20] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward. System Monitoring with Metric-Correlation Models: Problems and Solutions. In *ACM International Conference on Automatic Computing (ICAC)*, 2009
[21] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li. Mining Invariants from Console Logs for System Problem Detection. In *USENIX Annual Technical Conference (ATC)*, 2010

Instead of detecting violations on metric values, as all aforementioned approaches do, EbAT [22, 23, 24], NMI(Normalized Mutual Information) [25], and Parametric Mixture Distribution [26] use *information and probability theories* to detect anomalies in metric distributions. For instance, EbAT aggregates multiple metrics into a random variable, and tracks their distribution patterns at runtime. The anomalies are detected via identification of outliers in the distribution patterns.

[22] C. Wang, V. Talwar, K. Schwan, and P. Ranganathan. Online Detection of Utility Cloud Anomalies Using Metric Distributions. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010
[23] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical Techniques for Online Anomaly Detection in Data Centers. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011
[24] C. Wang. EbAT: online methods for detecting utility cloud anomalies. In *the 6th Middleware Doctoral Symposium (MDS 2009) in conjunction with Middleware 2009*, 2009
[25] M. Jiang, M. A. Munawar, T. Reidemeister, and P. A. Ward. Automatic Fault Detection and Diagnosis in Complex Software Systems by Information-Theoretic Monitoring. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2009
[26] K. Ozonat. An Information-Theoretic Approach to Detecting Performance Anomalies and Changes for Large-Scale Distributed Web Services. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2008

There is also research on post-detection, concerned with reducing false positives or false alarms and/or to prioritize the severity of anomalies. Examples include the anomaly-ranking approaches described in [27, 28].

[27] K. Viswanathan, L. Choudur, V. Talwar, C. Wang, G. MacDonald, and W. Satterfield. Ranking Anomalies in Data Centers. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2012
[28] G. Jiang, H. Chen, K. Yoshihira, and A. Saxena. Ranking the Importance of Alerts for Problem Determination in Large Computer Systems. In *ACM International Conference on Automatic Computing (ICAC)*, 2009

5.2 Proactive Detection

A shortcoming of reactive approaches is that they cannot prevent performance problems from occurring. Proactive approaches usually assume that there are existing measurements to describe normal or abnormal states of the system (KPIs or specific metrics and associated thresholds/patterns). They raise alarms when the system in still in normal state by looking at the predefined measurements, predicting when the problem will happen and on which machine.

[29] investigated the predictability of real-world systems and concluded that anomaly prediction approaches can be applied to them with high accuracy and significant lead time. [30] employed Bayesian classification methods and Markov models to predict impending anomalies in distributed data streaming applications on-line. ALERT [31] used an adaptive scheme to adjust the prediction models in different workload contexts, addressing the dynamism in date center environment. [32] proposed a integrated framework of measurement and system modeling techniques to detect application performance changes and differentiating

performance anomalies and workload changes.

[29] Y. Tan and X. Gu. On Predictability of System Anomalies in Real World. In *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010
[30] X. Gu and H. Wang. Online Anomaly Prediction for Robust Cluster Systems. In *IEEE International Conference on Data Engineering (ICDE)*, 2009
[31] Y. Tan, X. Gu, and H. Wang. Adaptive System Anomaly Prediction for Large-Scale Hosting Infrastructures. In *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2010
[32] L. Cherkasova, K. M. Ozonat, N. Mi, J. Symons, and E. Smirni. Anomaly? Application Change? or Workload Change? Towards Automated Detection of Application Performance Anomaly and Change. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2008

6. DIAGNOSIS

6.1 Dependency Inference

An important topic in diagnosis is to understand the spatial or temporal relationships between software or hardware components in enterprise data centers. *Dependency inference* analyzes relationships between interconnected components for problem troubleshooting, particularly when localizing the causes of problems that propagate across a distributed system. Specific instances of insights sought by dependency inference include service dependencies, request or call paths and transaction tracking through a distributed system.

Dependencies can be described at different levels, leading to different inference approaches. Orion [33], Project5 [34], Sherlock [35], ADD (Active Dependency Discovery) [36], and E2EProf [37] infer machine level dependencies.

[33] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating Network Application Dependency Discovery: Experiences, Limitations, and New Solutions. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008
[34] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed system of black boxes. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003
[35] P. Bahl, R. Chandra, A. G. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards Highly Reliable Enterprise Network Services via Inference of Multi-level Dependencies. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2007
[36] A. Brown, G. Kar, and A. Keller. An Active Approach to Characterizing Dynamic Dependencies for Problem Determination in a Distributed Environment. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2001
[37] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2EProf: Automated End-to-End Performance Management for Enterprise Systems. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2007

For request level diagnosis, Spectroscope [38] is a diagnosis approach that identifies and ranks changes in the flow and/or the timing of request processing, by comparing requests flows from two executions of the same application. X-ray [39] diagnoses the root causes of performance problems by instrumenting binaries as applications execute, and by using dynamic information flow tracking to estimate the likelihood that a block was executed due to each potential root cause. X-ray also highlights performance differences between two similar activities by differentially comparing their request execution paths.

[38] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing Performance Changes by Comparing Request Flows. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011
[39] M. Attariyan, M. Chow, and J. Flinn. X-ray: Automating root-cause diagnosis of performance anomalies in production software. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012

State-transition-based Transaction Tracking [40], CAG [41], vPath [42], and NetMedic [43] track process or thread level dependencies.

[40] A. Anandkumar, C. Bisdikian, and D. Agrawal. Tracking in a Spaghetti Bowl: Monitoring Transactions Using Footprints. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2008
[41] Z. Zhang, J. Zhan, Y. Li, L. Wang, D. Meng, and B. Sang. Precise Request Tracing and Performance Debugging for Multi-Tier Services of Black Boxes. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2009
[42] B. C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang. vpath: precise discovery of request processing paths from black-box observations of thread and network activities. In *USENIX Annual Technical Conference (ATC)*, 2009
[43] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl. Detailed diagnosis in enterprise networks. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2009

Active Probing [44] uses an incremental approach to infer service dependencies via active probing. Pranaali [45] also infers relationships and partitions between subcomponents of services. MonitorRank [46] periodically generates service dependencies which are then leveraged by unsupervised machine learning models for suggesting root cause candidates.

[44] I. Rish, M. Brodie, N. Odintsova, S. Ma, and G. Grabarnik. Real-Time Problem Determination in Distributed Systems Using Active Probing. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2004
[45] V. Kumar, K. Schwan, S. Iyer, Y. Chen, and A. Sahai. A State-Space Approach to SLA Based Management. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2008
[46] M. Kim, R. Sumbaly, and S. Shah. Root Cause

Detection in a Service-Oriented Architecture. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*

Researchers use a variety of techniques for inference. Orion [33], Project5 [34], and E2EProf [37] use network traffic and signal processing methods to infer dependencies without detailed knowledge about the application being observed, so that they can operate in the 'black box' settings experienced in virtualized systems. Sherlock [35] and NetMedic [43] leverage application-level knowledge such as configuration information and historical failure/success records, along with network traffic information, to infer dependencies. CAG [41] and vPath [42] use system- or kernel-level traces to infer dependencies, while X-ray [39] uses binary instrumentation to infer application-level dependencies.

State-transition-based Transaction Tracking [40] uses Markov Chain models along with the ARM instrumentation metrics to infer the dependencies of transaction footprints. ADD (Active Dependency Discovery) [36] finds dynamic dependencies by systematically perturbing the system, which is also used in the Bayesian modeling based approach in [45] to first partition a larger scale system into different state spaces, each of which can then be investigated separately.

6.2 Correlation Analysis

Correlation analysis explores the correlations between the observed metrics and the problematic/normal behaviors of data centers, or it considers correlations between metrics to characterize the patterns of the system, (*e.g.*, clustering metrics in some normal system state to construct a model of such states). Most correlation analysis approaches use machine learning techniques to correlate metrics to known problems.

Signature [47, 48], HPD (Heath Problem Detection) [49], Symptom-Matching [50], Fingerprint [51], iManage [52], Pinpoint [53], and Trace-based Problem Diagnosis [54] correlate SLA violations or known problems to system-level metrics. [55] correlates system changes to the failure or success symptoms of the system. CAL [8] correlates a request path to its failure or success states using decision tree methods. Draco [56] uses Bayesian analysis to localize chronic problems by comparing the distribution of features in successful and failed requests. [14] and Fa [57] build correlation models among monitoring data in normal state by using clustering methods.

[47] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, Indexing, Clustering, and Retrieving System History. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005

[48] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating Instrumentation Data to System States: a Building Block for Automated Diagnosis and Control. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004

[49] M. K. Agarwal, M. G. 0007, V. Mann, N. Sachindran, N. Anerousis, and L. B. Mummert. Problem Determination in Enterprise Middleware Systems using Change Point Correlation of Time Series Data. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006

[50] M. Brodie, S. Ma, G. Lohman, L. Mignet, N. Modani, M. Wilding, J. Champlin, and P. Sohn. Quickly Finding Known Software Problems via Automated Symptom Matching. In *ACM International Conference on Automatic Computing (ICAC)*, 2005

[51] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen. Fingerprinting the Datacenter: Automated Classification of Performance Crises. In *European Conference on Computer Systems (EuroSys)*, 2010

[52] V. Kumar, B. F. Cooper, G. Eisenhauer, and K. Schwan. iManage: Policy-Driven Self-Management for Enterprise-Scale Systems. In *ACM/IFIP/USENIX International Conference on Middleware*, 2007

[53] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2002

[54] C. Yuan, N. Lao, J.-R. Wen, J. Li, Z. Zhang, Y.-M. Wang, and W.-Y. Ma. Automated Known Problem Diagnosis with Event Traces. In *European Conference on Computer Systems (EuroSys)*, 2006

[55] M. Agarwal and V. Madduri. Correlating Failures with Asynchronous Changes for Root Cause Analysis in Enterprise Environments. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2010

[8] M. Y. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. A. Brewer. Failure Diagnosis Using Decision Trees. In *ACM International Conference on Automatic Computing (ICAC)*, 2004

[56] S. P. Kavulya, S. Daniels, K. R. Joshi, M. A. Hiltunen, R. Gandhi, and P. Narasimhan. Draco: Statistical diagnosis of chronic problems in large distributed systems. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2012

[14] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2009

[57] S. Duan and S. Babu. Guided Problem Diagnosis through Active Learning. In *ACM International Conference on Automatic Computing (ICAC)*, 2008

6.3 Similarity Analysis

Peer Comparison [58], Ganesha [59], Kahuna [60], and PeerWatch [18] detect and localize problems by comparing behaviors of peer machines or peer software components, under the assumption that in normal state the peers should perform similarly. Hence, the outliers deviating from the similarity are the problematic nodes or components. Peer-comparison approaches assume that the majority of peers in the system are fault-free.

[58] M. P. Kasick, J. Tan, R. Gandhi, and P. Narasimhan. Black-box Problem Diagnosis in Parallel File Systems. In *USENIX Conference on File and Storage Technologies (FAST)*, 2010

[59] X. Pan, J. Tan, S. Kavulya, G. R., and P. Narasimhan. Ganesha: Black-box diagnosis of mapreduce systems. In *Workshop on Hot Topics in Measurement and Modeling of Computer Systems (HotMetrics)*, 2009

[60] J. Tan, X. Pan, E. Marinelli, S. Kavulya, R. Gandhi, and P. Narasimhan. Kahuna: Problem diagnosis for mapreduce-based cloud computing environments. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*,

6.4 Detection vs. Diagnosis

Detection and diagnosis are typically applied to two different phases of performance troubleshooting procedure. For instance, detailed diagnosis of root-causes usually takes place after alarms are raised by anomaly detection. Based on the nature of the their respective timings, a detection approach should primarily consider responsiveness while diagnosis should emphasize comprehensiveness and detailedness. That's why we can find that current detection methods are usually on-line and apply to small volume, real-time monitoring metrics while diagnosis approaches are usually off-line and analyze large volumes of historical monitoring data.

However, above differences are not decisive. Some diagnosis approaches such as Kahuna [60] can do on-line diagnosis while some detection approaches like [14] need off-line analysis on history data to build a model beforehand for real-time anomaly detection. In this sense, the difference between detection and diagnosis is blurring.

7. SUPPORTING INFRASTRUCTURES

7.1 Monitoring Infrastructures

Monitoring systems are the foundation for performance troubleshooting in data centers — their purpose being to keep track of the 'health' status of the whole data center and/or the end-to-end performance of the applications. As a result, scalability, on-line operation, and low overheads that minimally perturb running applications are the dominant design criteria.

Monitoring infrastructures can be categorized into state monitoring vs. aggregation systems. Ganglia [61], Nagios [62], REMO [63], Query [64], CoMon [65], and foTrack [66] are *state monitoring systems* that are able to track overall system states in large scale data centers, ranging from thousands of nodes (as in Ganglia) to 60,000+ nodes (as in Query). These systems typically use simple analysis functions like thresholding, and there has been research [63, 66] to optimize analysis algorithms to reduce false positives and overheads in terms of resource consumption for monitoring.

[61] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation And Experience. *Parallel Computing*, 2003
[62] E. Galstad. Nagios Enterprises LLC., 2008
[63] S. Meng, S. Kashyap, C. Venkatramani, and L. Liu. REMO: Resource-Aware Application State Monitoring for Large-Scale Distributed Systems. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2009
[64] T. Repantis, J. Cohen, S. Smith, and J. Wein. Scaling a Monitoring Infrastructure for the Akamai Network. *SIGOPS Operating Systems Review*, 2010
[65] K. Park and V. S. Pai. CoMon: a Mostly-Scalable Monitoring System for PlanetLab. *SIGOPS Operating Systems Review*, 2006
[66] Y. Zhao, Y. Tan, Z. Gong, X. Gu, and M. Wamboldt. Self-Correlating Predictive Information Tracking for Large-Scale Production Systems. In *ACM International Conference on Automatic Computing (ICAC)*, 2009

Aggregation systems collect and summarize large volumes of monitoring data distributed among data center nodes, to yield a global summary of system health. Astrolabe [67], SDIMS [68], Moara [69], San Fermin [70], and Network Imprecision [71] are representative systems in this category.

[67] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Transactions on Computer Systems*, 2003
[68] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. In *ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2004
[69] S. Y. Ko, P. Yalagandula, I. Gupta, V. Talwar, D. Milojicic, and S. Iyer. Moara: Flexible and Scalable Group-Based Querying System. In *ACM/IFIP/USENIX International Conference on Middleware*, 2008
[70] J. Cappos and J. H. Hartman. San Fermin: Aggregating Large Data Sets Using a Binomial Swap Forest. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008
[71] N. Jain, P. Mahajan, D. Kit, P. Yalagandula, M. Dahlin, and Y. Zhang. Network Imprecision: a New Consistency Metric for Scalable Monitoring. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008

7.2 Tracing Infrastructures

The typical purpose of tracing infrastructures is to support the diagnosis phase of performance troubleshooting. They provide mechanisms to trace events both along (*i.e.*, horizontally) and across (*i.e.*, vertically) the software stack in data center systems. Data may be collected at hypervisor, kernel, middleware, and application levels, and such data typically consists of extensive trace logs that are not typically seen in the monitoring systems described in Section 7.1 because of the potentially high overheads or perturbation associated with their collection and use. Research in this space, therefore, is often concerned with (1) how to reduce tracing overhead and perturbation, (2) how to easily and effectively instrument systems and applications, and (3) how to carry out on-line tracing at acceptable cost.

Fay [72], Chopstix [73], Dapper [74], and GWP (Google-Wide Profiling) [75] use sampling techniques to trace its large scale data center on-line with high effectiveness in problem detection and diagnosis.

[72] U. Erlingsson, M. Peinado, S. Peter, and M. Budiu. Fay: Extensible Distributed Tracing from Kernels to Clusters. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011
[73] S. Bhatia, A. Kumar, M. E. Fiuczynski, and L. Peterson. Lightweight, High-Resolution Monitoring for Troubleshooting Production Systems. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2008
[74] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure. Technical report, Google, 2010
[75] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt. Google-Wide Profiling: A Continuous Profiling

Infrastructure for Data Centers. *Micro, IEEE*, 2010

Whodunit [76], DARC [77], Stardust [78], X-Trace [79], D^3S [80], and Chirp [81] provide systems or programming tools to instrument data center applications or systems for tracing and querying their runtime states. SysProf [82] uses kernel level instrumentation to realize fine-grained, low-overhead tracing. Pip [83] is an instrumentation framework that allows programmers to embed expectations about a system's communications structure, timing and resource consumption. Pip detects problems by comparing actual behavior against expected behavior.

[76] A. Chanda, A. L. Cox, and W. Zwaenepoel. Whodunit: Transactional Profiling for Multi-Tier Applications. In *European Conference on Computer Systems (EuroSys)*, 2007

[77] A. Traeger, I. Deras, and E. Zadok. DARC: Dynamic Analysis of Root Causes of Latency Distributions. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2008

[78] E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez, and G. R. Ganger. Stardust: Tracking Activity in a Distributed Storage System. *SIGMETRICS Performance Evaluation Review*, 2006

[79] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-Trace: a Pervasive Network Tracing Framework. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007

[80] X. Liu, Z. Guo, X. Wang, F. Chen, X. Lian, J. Tang, M. Wu, M. F. Kaashoek, and Z. Zhang. D^3S: Debugging Deployed Distributed Systems. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008

[81] E. Anderson, C. Hoover, X. Li, and J. Tucek. Efficient Tracing and Performance Analysis for Large Distributed Systems. In *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2009

[82] S. Agarwala and K. Schwan. SysProf: Online Distributed Behavior Diagnosis through Fine-Grain System Monitoring. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2006

[83] P. Reynolds, C. Killian, J. L. Wiener, J. C. Mogul, M. A. Shah, and A. Vahdat. Pip: Detecting the Unexpected in Distributed Systems. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2006

7.3 Analytics Infrastructures

Analytics infrastructures support performance troubleshooting by making it easy to apply various analytics functions to monitoring metrics in data center. Like data mining systems, their focus is data analysis rather than the data collection actions taken by tracking and monitoring infrastructures though some of these analytics systems, however, have associated data monitoring and tracing functions. Magpie [84] is a system for tracing and also clustering requests to learn about interaction patterns in the system being observed. SelfTalk [85] is a query system supporting various queries useful for understanding system behaviors of multi-tier web applications. Monalytics [86, 87] combines monitoring and existing analytics approaches (detection, diagnosis approaches) to offer a flexible architecture by which a wide range of analytics can be applied to various monitoring metrics/traces on any node in data centers. Monalytics was built on the vManage [88] architecture designed for managing virtualized data centers. VScope [89] is a flexible middleware for troubleshooting performance problems in large scale real-time big data applications.

[84] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for Request Extraction and Workload Modelling. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004

[85] S. Ghanbari, G. Soundararajan, and C. Amza. A Query Language and Runtime Tool for Evaluating Behavior of Multi-Tier Servers. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2010

[86] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf. Monalytics: Online Monitoring and Analytics for Managing Large Scale Data Centers. In *ACM International Conference on Automatic Computing (ICAC)*, 2010

[87] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, and M. Wolf. A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-Scale Data Centers. In *ACM International Conference on Automatic Computing (ICAC)*, 2011

[88] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan. vManage: Loosely Coupled Platform and Virtualization Management in Data Centers. In *ACM International Conference on Automatic Computing (ICAC)*, 2009

[89] C. Wang, I. A. Rayan, G. Eisenhauer, K. Schwan, V. Talwar, M. Wolf, and C. Huneycutt. VScope: Middleware for Troubleshooting Time-Sensitive Data Center Applications. In *ACM/IFIP/USENIX International Conference on Middleware (Middleware)*, 2012

8. REMEDIATION

As stated earlier, a wide variety of techniques and approaches are used to mitigate or remedy problems. Below, we only review the representative recent work focused on data center systems and applications.

The TCP splice-based web server [90], SLACH [91], and the data recovery system [92] study novel system designs to support fault tolerance in enterprise data center applications. NAP [93], Policy Refinement [94], QoS-aware fault tolerant middleware [95], and [96] specialize in planning and optimizing recovery policies to best improve system performance. RobustStore [97], FATE and DESTINI [98], RFD analysis models [99], and the analytic queuing models used in [100] are tools and models that analyze the effects of recovery actions on applications.

[90] M. Marwah, S. Mishra, and C. Fetzer. Enhanced Server Fault-Tolerance for Improved User Experience. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2008

[91] J. Zhou, C. Zhang, H. Tang, J. Wu, and T. Yang. Programming Support and Adaptive Checkpointing for High-Throughput Data Services with Log-Based Recovery. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2010

[92] I. Akkus and A. Goel. Data Recovery for Web Applications. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2010

[93] M. Ben-Yehuda, D. Breitgand, M. Factor, H. Kolodner,

V. Kravtsov, and D. Pelleg. NAP: a Building Block for Remediating Performance Bottlenecks via Black Box Network Analysis. In *ACM International Conference on Automatic Computing (ICAC)*, 2009
[94] M. Goldszmidt, M. Budiu, Y. Zhang, and M. Pechuk. Toward Automatic Policy Refinement in Repair Services for Large Distributed Systems. *SIGOPS Operating Systems Review*, 2010
[95] Z. Zheng and M. Lyu. A QoS-Aware Fault Tolerant Middleware for Dependable Service Composition. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2009
[96] Q. Zhu and C. Yuan. A Reinforcement Learning Approach to Automatic Error Recovery. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2007
[97] L. Buzato, G. Vieira, and W. Zwaenepoel. Dynamic Content Web Applications: Crash, Failover, and Recovery Analysis. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2009
[98] H. S. Gunawi, T. Do, P. Joshi, P. Alvaro, J. M. Hellerstein, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, K. Sen, and D. Borthakur. FATE and DESTINI: a Framework for Cloud Recovery Testing. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2011
[99] T. Ma, J. Hillston, and S. Anderson. On the Quality of Service of Crash-Recovery Failure Detectors. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2007

Recovery actions most commonly use rebooting for failure recovery or performance remediation, with improvements sought by methods like Micro-reboot [101], JAGR [102], RETRO [103], Stochastic Reward Nets(SRNs), and Software Rejuvenation [104].

[100] H. P. Schwefel and I. Antonios. Performability Models for Multi-Server Systems with High-Variance Repair Durations. In *IEEE Conference on Dependable Systems and Networks (DSN)*, 2007
[101] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot: A technique for cheap recovery. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004
[102] G. Candea, E. Kiciman, S. Kawamoto, and A. Fox. Autonomous Recovery in Componentized Internet Applications. *Cluster Computing*, 2006
[103] T. Kim, X. Wang, N. Zeldovich, and M. F. Kaashoek. Intrusion Recovery Using Selective Re-Execution. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010
[104] K. Vaidyanathan, R. E. Harper, S. W. Hunter, and K. S. Trivedi. Analysis and implementation of software rejuvenation in cluster systems. In *ACM Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2001

In recent big-data frameworks, such as MapReduce, performance problems are addressed by rescheduling slow tasks, and intelligent data placement. Mantri [105] uses statistical tests to detect 'straggler' tasks caused by data skew, network congestion and machine failures in large-scale MapReduce jobs. Mantri takes the appropriate recovery action, e.g., task re-execution, based on the underlying cause. Scarlett [106] tackles the file system performance problems in MapReduce clusters by predicting the data 'hotspots' and replicating data blocks based on their popularity. In virtualized data center scenario, Net-cohort [107] leverages clustering techniques to discover VM cohort which is a group of VMs working closely to each other. By migrating VMs, which belong to the same cohort, to the same physical host, Net-cohort can effectively relieve the resource bottleneck (e.g. cross-rack bandwidth contention) and thereby improve the performance of big data applications running in the data center.

[105] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, B. Saha, and E. Harris. Reining in the Outliers in Map-Reduce Clusters Using Mantri. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010
[106] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, and E. Harris. Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters. In *European Conference on Computer Systems (EuroSys)*, 2011
[107] L. Hu, K. Schwan, A. Gulati, J. Zhang, and C. Wang. Net-Cohort: Detecting and Managing VM Ensembles in Virtualized Data Centers. In *ACM International Conference on Automatic Computing (ICAC)*, 2012

There is much valuable, previous and ongoing research in high reliability systems obtained through redundancy and replication [108], which in turn builds on extensive prior work in reliability study. In data center systems' 'scale-out' infrastructures, this has led to the common use of data replication, fail-over servers, quorums, and the many associated methods for fault tolerant, high availability operation. We do not review such work here, pointing the reader at previous summaries for research like [109].

[108] M. Hiltunen, R. Schlichting, and C. Ugarte. Building Survivable Services Using Redundancy and Adaptation. *Computers, IEEE Transactions on*, 2003
[109] R. Guerraoui and A. Schiper. Software-Based Replication for Fault Tolerance. *Computer*, 1997